Lernzettel

Asynchrone Kommunikation und Messaging-Patterns: Publish/Subscribe, Message Queues, Event Sourcing, eventual consistency

Universität:	Technische Universität Berlin
Kurs/Modul:	Architektur von Anwendungssystemen
Erstellungsdatum:	September 19, 2025

Lernzettel: Asynchrone Kommunikation und Messaging-Patterns: Publish/Subscribe, Message Queues, Event Sourcing, eventual consistency

(1) Überblick und Lernzielrichtung. Dieser Lernzettel behandelt zentrale Muster der asynchronen Kommunikation in verteilten Anwendungen: Publish/Subscribe, Message Queues, Event Sourcing und eventual consistency. Ziel ist es, die Entkopplung, Skalierbarkeit und die Auswirkungen auf Konsistenz zu verstehen und typische Einsatzszenarien sowie Herausforderungen zu erkennen.

(2) Publish/Subscribe (Pub/Sub).

- Aufbau: Publisher, Subscriber(n), Broker bzw. Message Bus, Topic(s).
- Ablauf: Ein Publisher sendet eine Nachricht an ein Topic; der Broker verteilt die Nachricht an alle Abonnenten des Topics.
- Eigenschaften: Lose Kopplung zwischen Publishern und Abonnenten, asynchrone Verarbeitung, Skalierbarkeit durch verteilte Broker.
- Typische Einsatzfälle: Benachrichtigungen, Streaming von Events, Logging- und Audit-Streams.
- Herausforderungen: Reihenfolge ist brokerabhängig; Duplicate-Delivery möglich; oft eventual oder bestimmte Exactly-Once-Garantien erfordern idempotente Verarbeitung.

(3) Message Queues (MQ).

- Muster: Point-to-Point (eine Nachricht, ein Verbraucher) vs. Publish/Subscribe über Queues.
- Bestandteile: Queue, Producer, Consumer, Acknowledge (Ack), Dead-Letter-Queue (DLQ).
- Semantiken: At-Least-Once, At-Most-Once, Exactly-Once (je nach Implementierung).
- Vorteile: Zuverlässige asynchrone Verarbeitung, Laststeuerung, Pufferung von Spitzenlasten.
- Herausforderungen: Konsistenz über verteilte Services, Message-Duplicate-Handling, State-Management außerhalb der Queue.

(4) Event Sourcing.

- Grundidee: Alle Änderungen des Anwendungszustands werden als immutable Events in einer Event-Log-Datei gespeichert.
- Zustand rekonstruieren: Der aktuelle State ergibt sich aus dem Replaying der Events in Reihenfolge.
- Vorteile: Vollständiges Audit-Trail, Reproduzierbarkeit, einfache Unterstützung von Undo/Replay, einfache Skalierbarkeit von Schreibzugriffen.
- Herausforderungen: Event-Schema-Evolution und Versionierung, Snapshotting zur Performance, Migration alter Events, Storage-Management und Abwärtskompatibilität.

(5) Eventual Consistency.

- Definition: Das Gesamtsystem konvergiert schließlich zu einem konsistenten Zustand, auch wenn einzelne Knoten zeitweise inkonsistent sind.
- Zusammenhang: Oft in verteilten Systemen bei asynchroner Kommunikation; Trade-off zwischen Verfügbarkeit, Partitionstoleranz und Konsistenz (CAP-Theorem).
- Typische Modelle und Konzepte: Read-your-writes, causale Konsistenz, eventual guarantees, Konfliktauflösung bei Replikationen.
- Praktische Folgen: Mögliche Sicht auf veraltete Daten, Notwendigkeit idempotenter Verarbeitung, Konfliktlösung bei gleichzeitigen Änderungen.

(6) Architektur-Pattern-Vergleich (Tendenzen).

- Pub/Sub eignet sich gut für Notification- und Streaming-Szenarien, fokussiert auf Entkopplung und Skalierbarkeit.
- MQ eignet sich gut für zuverlässige Hintergrundaufgaben, Job-Steuerung, Lastverteilung und kontrollierte Verarbeitung mit Ack-Logik.
- Event Sourcing bietet starke Auditierbarkeit und Rekonstruktion des Zustands, ist aber komplexer in der Evolution des Event-Modells.
- Eventual Consistency passt gut zu stark verteilten Systemen mit asynchronen Flows, verlangt aber geeignete Konfliktlösungen und klare Semantik der Reads.

(7) Praxisbeispiele: einfache Architekturen.

- Beispiel 1 – Bestellabwicklung: Ein Order-Service veröffentlicht Events (Bestellung erstellt, Zahlung bestätigt) über Pub/Sub; ein Inventory-Service und ein Shipping-Service hören die relevanten Events und verarbeiten asynchron.
- Beispiel 2 – Hintergrundverarbeitung: Aufgaben werden in einer MQ-Warteschlange abgesetzt; Worker-Instanzen entkoppeln die Verarbeitung, verwenden Dead-Letter-Queues für Fehlerfälle.
- Beispiel 3 – Event Styles: Zusätzlich zum aktuellen Zustand werden Events genutzt, um Berichte zu erzeugen oder Zustandsveränderungen über Replay nachvollziehbar zu machen (Event Sourcing).

(8) Übungs-/Checkliste.

- Welche Muster eignen sich für einfache Benachrichtigungen vs. komplexe Geschäftsprozesse?
- Wie sicherstelle ich Idempotenz bei wiederholten Nachrichten?
- Welche Konsistenzgarantien bieten Pub/Sub vs. MQ in deiner Architektur?
- Wie lässt sich Event Sourcing sinnvoll mit Snapshotting kombinieren?

Zusammenfassung. - Pub/Sub, MQ, Event Sourcing und eventual consistency adressieren zentrale Aspekte asynchroner Architekturen: Entkopplung, Skalierbarkeit, Fehlertoleranz und konsistenzbezogene Kompromisse. - Die Wahl des Musters hängt von den Anforderungen an Konsistenz, Auditierbarkeit, Latenz und Komplexität ab. - Wichtig ist eine klare Semantik, idempotente Verarbeitung, geeignete Fehlerbehandlung und gegebenenfalls Snapshotting/Replay-Strategien.